ECE 661: Homework #3
RNN and Transformers

Hai Li

ECE Department, Duke University — February, 2025

Objectives

This assignment aims to provide hands-on experience with Recurrent Neural Networks (RNNs) and Large Language Models (LLMs) through two main labs:

Lab 1: Implement an LSTM model for sentiment analysis using the IMDB dataset. Students will build a data loader, create a vocabulary, implement tokenization, and construct an LSTM model for binary sentiment classification.

Lab 2: Explore Large Language Models using GPT-2. Students will generate text with a pre-trained model, prepare a dataset, evaluate model performance using perplexity, fine-tune the model, and compare full fine-tuning with LoRA (Low-Rank Adaptation) fine-tuning.

Through these labs, students will gain practical experience in natural language processing tasks, model implementation, and working with state-of-the-art language models.

Warning: You are asked to complete the assignment independently.

This lab has 100 points plus 10 bonus points, yet your final score cannot exceed 100 points. The submission deadline will be **11:55pm, Wednesday, March 5**. We provide a template named `LabRNN.ipynb` and `LabLLM.ipynb` to start with, and you are asked to develop your own code based on this template. You will need to submit two independent files including:

1. A self-contained PDF report, which provides answers to all the conceptual questions and clearly demonstrates all your lab results and observations and required code snippet. Remember, **do NOT generate PDF from your jupyter notebook to serve as the report**, which can increase the TA's burden of grading.
2. `code.zip`, a zipped code file which contains 2 jupyter notebooks `LabRNN.ipynb`, `LabLLM.ipynb`, respectively for the two labs.

Note that 20 percent of the grade will be deducted if the submissions doesn't follow the above guidance.

Note that TAs hold the right to adjust grading based on the returned homeworks. We make sure that the grading rule is consistent among all students. Also, the results given for the Labs (for example the reported accuracies) are obtained from the specific runtime when TAs were working on the answers. We do not expect you to get exactly the same numbers; yet, it is necessary that your results show the same trends/patterns/observations in order to receive full credits.

1 True/False Questions (30 pts)

For each question, please provide a short explanation to support your judgment or fix the wrong statement.

Problem 1.1 (3pts) Long short-term memory (LSTM) is able to configure how much the history information to keep. Particularly, the forget gate is designed to control how much of the previous cell state to retain. we cannot adjust the rest parameters within one LSTM cell to achieve identical effect as changing the forget gate.

Problem 1.2 (3 pts) In a simple self-attention, the time complexity is $O(n^2d)$ for query, key, and value (q, k, v) with sequence length n and dimensionality d. For multi-head attention, where the number of heads is h and the projections W^Q, W^K, $W^V \in \mathbb{R}^{d\times(d/h)}$, the time complexity remains $O(n^2d)$.

Problem 1.3 (3pts) Byte Pair Encoding (BPE) is a "Bottom-up" tokenization technique and is used in GPT and BERT. Compared to traditional word-level tokenization, it reduces the vocabulary size significantly and provides better handling of unknown words.

Problem 1.4 (3pts) In transformers, positional embedding is multiplied on the embedding of each token to represent the position relationship between tokens.

Problem 1.5 (3pts) In an encoder-decoder transformer model, the information captured by the encoder is used as the "Query" and "Value" in the second multi-head attention block in the decoder. (Hint: Refer to Lecture 10 slides).

Problem 1.6 (3pts) In LLMs, a token stands for a word in the input prompt or output prompt.

Problem 1.7 (3pts) Assume an LLMs input token length is N and generated token length is M and the hidden dimension is d. In decoding phase the computation complexity for computing Q,K,V in a layer of a single forward pass is $O((N + M)d)$.

Problem 1.8 (3pts) Flash attention is proposed to improve the computation locality of multihead attention computation in LLMs.

Problem 1.9 (3pts) In RLHF, human directly involve in the training process of LLMs, providing feedback to the LLM generated contents and train it accordingly.

Problem 1.10 (3pts) KV-Cache is not helpful in training large language model.

2 Lab (1): Recurrent Neural Network for Sentiment Analysis (35pts)

Google colab is a useful tool for modeling training. In order to run ipynb document, you need to upload your jupyter notebook document to google drive. Then double click the file in the google drive, which will lead you to google colab. Moreover, it'd be more efficient to use GPU to train your LSTM and Transformer models in this assignment. To use GPU, please check Runtime > Change run-time type, and select GPU as the hardware accelerator. You may click on side bar document icon to upload your dataset and python file there.

In this lab, you will learn to implement an LSTM model for sentiment analysis. Sentiment analysis [1, 2] is a classification task to identify the sentiment of language. It usually classifies data into two to three labels: positive, negative or/and neutral. IMDB [3] is one of the most widely used dataset for binary sentiment classification. It uses only positive and negative labels. IMDB contains 50,000movie review data collected from popular movie rating service IMDB. You may find more details at https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews.

To start this assignment, please open the provided Jupyter Notebook LabRNN.ipynb. We have provided implementation for the training recipe, including optimizers, learning rate schedulers, and weight initialization. You may NOT change any of the hyperparameter settings (i.e., the object HyperParams) in this lab.

Grading Instructions. For this part, we mainly grade on the logic and conciseness of the code. If a mistake is found in this part that lead to erroneous conclusions for questions in the later part, we will consider this and provide partial/full credit for the later part, to avoid applying the same penalty twice. **Please attach your added code in the report under corresponding question.** Note that TAs and Professor have the final discretion to adjust grade according to the given submission.

(a) (5 pts) Implement your own data loader function. First, read the data from the dataset file on the local disk. Then split the dataset into three sets: train, validation, and test by $7:1:2$ ratio. Finally return x_train, x_valid, x_test, y_train, y_valid and y_test, where x represents reviews and y represent labels.

(b) (5 pts) Implement the build_vocab function to build a vocabulary based on the training corpus. You should first compute the frequency of all the words in the training corpus. Remove the words that are in the STOP_WORDS. Then filter the words by their frequency ($\geq$ min_freq) and finally generate a corpus variable that contains a list of words.

(c) (5 pts) Implement the tokenization function. For each word, find its index in the vocabulary. Return a list of integers that represents the indices of words in the example.

(d) (5 pts) Implement the __getitem__ function in the IMDB class. Given an index i, you should return the i-th review and label. The review is originally a string. Please tokenize it into a sequence of token indices. Use the max_length parameter to truncate the sequence so that it contains at most max_length tokens. Convert the label string ('positive' / 'negative') to a binary index, such as 'positive' is 1 and 'negative' is 0. Return a dictionary containing three keys: 'ids', 'length', 'label' which represent the list of token ids, the length of the sequence, the binary label.

(e) (5 pts) In __init__, a LSTM model contains an embedding layer, an lstm cell, a linear layer, and a dropout layer. You can call functions from Pytorch's nn library. For example, nn.Embedding, nn.LSTM, nn.Linear...

(5 pts) In forward, decide where to apply dropout. The sequences in the batch have different lengths. Write/call a function to pad the sequences into the same length. Apply a fully-connected (fc) layer to the output of the LSTM layer. Return the output features which is of size [batch size, output dim].

(f) (5 pts) Train the LSTM model for 5 epochs with default configuration. Do you observe a steady and consistent decrease of the loss value as training progresses? Report your observation on the learning dynamics of training loss, and validation loss on the IMDB dataset. Besides, show the model prediction of the first test set. Do they meet your expectations and why?

3 Lab (2) Large Langugae Model for Text Generation (45 pts)

In this lab, we also recommend you using Google colab to finish the model training. You will learn to use hugging face transformers to run and fine-tune large language models. Please follow the instruction in jupyter notebook to finish the task and attach your implemented code in the report under the corresponding question.

(a) (10 pts) Load the pre-trained GPT-2 and generate some text. It is recommended to use .generate() for token generation and .decode to decode token into text.

(b) (10 pts) Prepare the dataset. We are using wiki-text as test and training dataset, but it is a relative large dataset, we are only using 10% of data for practice.

(c) (10 pts) Implement the inference of GPT-2, and evaluate its performance. We use perplexity as the evaluation metric, which is a common metric for LLMs.

(d) (5 pts) Finetune the GPT-2 model on wiki-text dataset. Show how the train and validation loss change in different epoch. Hugging face provide convenient function to train a transformers, there is no need to implement your own training function. After finetuning, load the finetuned model and evaluate the perplexity and try to generate some text.

(e) (5 pts) Use LoRA to finetune the GPT-2 model. Hugging face peft has implemented LoRA, directly call the function to realize LoRA finetuning.

(f) (5 pts) Compare the finetuning time of fully finetuning and LoRA finetuning, explain the reason. Besides, compare the perplexity and generated text of Pre-trained GPT-2 and finetuned, LoRA finetuned GPT-2, explain the reason.